Center for Advanced Computation

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

CAC Document 170
JTSA Document Number 5512

Research in Network Data Management and Resource Sharing

Preliminary Experimental System Design Report

August 20, 1975

CAC Document Number 170
JTSA Document Number 5512

Research in Network Data Management and Resource Sharing

Preliminary Experimental System Design Report

by

Steve R. Bunch
David C. Healy
Edwin J. McCauley

Prepared for the
Joint Technical Support Activity
of the
Defense Communications Agency
Washington, D.C.

under contract
DCA100-75-C-0021

Center for Advanced Computation
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

August 20, 1975

Approved for release: Peter A. Alsberg
Peter A. Alsberg, Principal Investigator

Table of Contents

Executive Summary

Background and Overview

This document presents the preliminary design of an experimental distributed data management system. The purpose of the system is to provide a vehicle for research in the areas of network data management and resource sharing for application to the World-Wide Military Command and Control System (WWMCCS) Intercomputer Network (WIN). The research is supported by the Joint Technical Support Activity of the Defense Communications Agency.

The design and implementation of the experimental system will serve as the focal point for the more pragmatic research topics in distributed data management, e.g. synchronization. While some of these topics may be theoretically understood, their implementation in a practical system poses major research challenges. When implemented, the system will provide a controlled environment for experiments. Presently, no data is available for many important parameters describing the operational characteristics of distributed data management systems. The experimental system must include a comprehensive and selective measurement capability so that these parameters may be ascertained.

To be used in the investigation of a range of topics, the system must be designed to be flexible and modular. From a discussion of the topics to be investigated via the experimental system and their relevance to the WWMCCS community, some of the necessary features of the experimental system are derived. Broadly speaking, the system design must not impose any artifical constraints on the experiments to be performed.

Nowhere is the requirement for flexibility more important than in the selection of the data model. The relational model captures the essence of other competing data models while permitting much greater freedom for experimentation. In addition, the relational model appears to be easier to implement and easier to explain to a user.

The choice of the machine on which to implement the system is most strongly influenced by the ease of creating experimental software. Multics represents perhaps the best system in existence for such an effort. Many of the most difficult aspects of system implementation, such as memory limitations and file management, are simply not problems on Multics. Since there are three Multics systems connected to the ARPA network, it will be possible to perform multi-machine experiments.

For the purposes of this document the system design has been broken down into three separate components: the network data manager, the local data manager and the network data base administrator support functions. The network data manager is responsible for coordinating the activities of the local data managers to answer a user query. The local data manager is responsible for retrieving, qualifying and sorting the data at his site and returning it to the network data manager. The network data base administrator support functions are the "staff" activities of the system (as contrasted with the "line" activities of the network and local data managers). To have an operational distributed data management system, the network data base administrator must be provided with a data collection and analysis capability. In the longer term many of the control loops which now include the data base administrator would be closed by having the system itself make some of the more routine decisions about restructuring, clustering, indexing, etc.

It should be emphasized at the outset that the experimental system must incorporate the results of a variety of different research topics. Because many of these topics represent areas of continuing research, no attempt is made to present definitive solutions here. The major topics for continued research are briefly sketched in this document. For a more detailed presentation of the topics and their interaction please see the Research Plan (CAC Doc. No. 164, JTSA Doc. No. 5510).

It is expected that approximately 50 man-months will be devoted to the design and initial implementation of the experimental system over the next 13.3 months of the follow-on contract. However, difficulties or breakthroughs in the supporting research could substantially alter these estimates.

Related Work

The project has produced ten other documents in the course of this research. Table 1 summarizes these other documents.

CAC Document Number	JTSA Document Number	Title	Date
149	5502	An Annotated Bibliography to Network Data Management and Related Literature	April 1975
150	5503	A State-of-the-Art Report on Network Data Management and Related Technology	April 1975
151	5504	WWMCCS System Summaries	April 1975
152	5505	Survey Report	April 1975
159	5506	Scenario Report	May 1975
160	5507	Application Summary	May 1975
161	5508	Technology Summary	May 1975
162	5509	Preliminary Research Study Report	May 1975
164	5510	Research Plan	June 1975
169	5511	Initial Mathematical Model Report	August 1975

Table 1

Supporting Documents

Although the reader would be well served by having some familiarity with these other documents, this familiarity will not be heavily relied upon here. The first two documents, the Annotated Bibliography and the State-of-the-Art Report are relatively comprehensive treatments of the existing literature and on-going activity in the field. The reader interested in a more thorough treatment of the concepts presented here is referred to those documents.

Areas of Experimentation

Introduction

The WWMCCS Intercomputer Network (WIN) is a natural candidate for distributed data management. By providing a distributed data management capability, it appears possible to achieve gains in system responsiveness, throughput and reliability.

Before proceeding with a WIN distributed data management system, many concepts must be evaluated and experiments must be performed to provide the technical base for the design of a production system. In this section we discuss the concepts to be evaluated and the experiments that may be performed with the experimental system.

Many of the experimental areas addressed have impact on both WIN and conventional, single-site data management systems. Where appropriate, this impact on both multi-site and single-site problems is discussed.

Reliability Improvements Through Automated Backup

A distributed data management system can offer improved reliability by duplication of critical data bases at multiple hosts. When the primary copy is unavailable (host goes down, disc drive fails, etc.) user operations can be automatically re-directed to a backup copy. When the problem is remedied, the original primary copy can be reinstated. These transitions (primary-to-backup and backup-to-primary) must be carefully designed and analyzed, lest problems such as critical races, deadlocks, or lost updates occur. In the Preliminary Research Study Report (CAC Doc. No. 162, JTSA Doc. No. 5509) one possible scheme was presented in detail. The Scenario Report (CAC Doc. No. 159, JTSA

Doc. No. 5506) presents a less detailed treatment of some other basic alternatives for reliability enhancement in a distributed data base. These are analyzed in more detail in the Initial Mathematical Model Report (CAC Doc. No. 169, JTSA Doc. No. 5511).

In a 35-node WIN, a host's going down is likely to be a frequent occurence. Thus a reliability improvement scheme using automated backup and recovery is essential. End-users should not be concerned with the mechanical aspects of backup and recovery. They should not be burdened with the necessity to program around failures of the primary copy or even of backup copies. Extensive operator intervention is similarly undesirable. In a 35-node WIN, with only one failure per host per day, every 40 minutes or so a host will fail somewhere. If operators throughout the network must manually switch to backup copies of data bases, they may wind up doing little else but this switching.

Such reliability improvements are not without cost. For example, all updates must be performed once for each copy of the data base. This can add significantly to processing loads. Additional problems exist due to the necessity of synchronization to insure data base consistency.

The experimental system should be capable of examining various backup and recovery strategies. It must also be capable of controlled simulation of a host failure, since the actual system can hardly be expected to fail at a convenient time.

Data Sharing and Concurrent Use

A distributed data base provides an inherent data sharing capability. Data maintained at one host can be easily accessible to suitably authorized users at other hosts. Sharing can be enhanced by a single user interface to the distributed data base system. The cumbersome aspects of TELNET-ing to another host, logging in there, and dealing with a foreign data base would then be eliminated. The WIN has a headstart in this area over research and commercial networks because of the standard systems used at each site and the commonality of interest and mission on the WIN.

Obviously, data sharing can be enhanced through concurrent use of the data base. The problem is to guarantee that the various concurrent users of a given data base do not interfere with one another. A sufficient (but not necessary) condition for non-interference is that only one user at a time may modify the data base. While this approach produces correct results, response time for queries is severely degraded as the number of updates increases. In the WIN environment many large data bases have small sections in which most of the activity is concentrated. For example, in FORSTAT less than 5% of the data is changed per day and only about 5% of the data changes over a multi-day period. If it were necessary to lock the entire data base every update, intolerable response degradation would result. Thus, a refinement of sharing through concurrent use would be to break the data base up into smaller, more manageable portions, along the conceptual lines of the DBTG* AREA concept discussed later. Then many concurrent users could operate on some data base sections while other sections were being modified. However this technique has the problem of increased cost and complexity since more objects must be managed. Deadlock is also possible.

---

* DBTG is the Data Base Task Group of CODASYL.

The experimental system should be capable of providing facilities for concurrent use of the data base to enhance data sharing. This necessitates an effective synchronization mechanism. Research into synchronization techniques for distributed data management systems is a top priority item.

Parallelism

The multiple host processors in a distributed data base system offer the possibility of parallelism. Parallelism can greatly improve the responsiveness of the system. It is in this area that a distributed data base system offers the most startling contrasts with existing, single-site systems. The basic idea is to perform activities in parallel on different hosts. For example, a commonly used component of many JOPS exercises is a large (300 Megabyte) airfield characteristics file. Currently it is a tedious and time-consuming task to process the score of tapes containing the information. If the file were suitably partitioned and located at different hosts, the searching could be accomplished in parallel in a fraction of the time it now takes. It may also be possible to exploit the network's parallelism when processing a query which accesses several files. A different host could be chosen to process part of the query for each different file in the query. Then the separate files could possibly all be processed in parallel.

The communication delay to ship large amounts of data across a network may well erase any advantages of parallel searching. Considerable study must go into the design of systems to take maximum advantage of the multiple hosts available. The obvious ideal situations are when large amounts of data must be searched for a few hits, i.e., needle in a haystack problems. Real world situations are not likely to be this ideal. In fact, we may not know in advance how many needles there are in the stack. If there are too many hits, the limiting factor becomes network bandwidth rather than host searching speed, and parallelism does not help much.

The ability to study the effects of alternative strategies for exploiting network parallelism is a central concept in the experimental system design. It is likely to be non-trivial to implement even a simple-minded scheme. A user request must be analyzed for the possibility of parallel operation. The advisability of actually performing the operations in parallel must be assessed. Finally, a complex multi-machine task must be coordinated and controlled in a hostile environment where hosts and communication links may fail at the most inappropriate times.

Load Balancing

By attempting to even out the load imposed on the hosts, a distributed data base system can give far better equipment utilization for the network as a whole. Load balancing is related to parallelism. Both concepts utilize a computer network as a network of cooperating hosts, not simply a disjoint collection of hosts with cheap phone lines to the users. The eventual exploitation of the WIN resource sharing capabilities is a primary goal of the entire research effort.

In the present WWMCCS environment, some sites are already incapable of handling their peak loads. A data base user at such a site may be given poor response or even denied service. A distributed data base system would allow the requests from such users to be handled at other sites if there were unused capacity. The user gets his job done faster, and excess capacity at under-utilized sites is better employed.

Ideally, the distributed data management system would determine a strategy for responding to a user's request in a minimum time. This would entail trading off the improved response of a remote host against the communication delays to send the request and receive the reply. An important area of research is the estimation of load-related factors like throughput and response time from observable indicators. Initial experiments on the ARPA network have shown that this estimation is a difficult task.

Conformal Data Structures

A distributed data base system should allow different data structures for different copies of the same data base located at different hosts (i.e. conformal structures). Copies could be structured to be optimized for a particular class of operations. For example, consider a file used in different ways by several groups. If multiple copies of the file (or portions of the file) exist, each copy can be physically structured in a different way. The distributed data base system would decide which copy has the most appropriate physical structure for each request received. The intent of multiple structures is to minimize the I/O and processing required to reply to a query. However, structures which are optimal for one collection of queries may be decidedly sub-optimal for a different class. As long as each of the copies continues to represent the same information, they remain suitable as backups, so reliability enhancement is still obtained. The question of how each copy should be structured is, of course, of major interest to the network data base administrator. If the secondary, derived data structures such as indices and saved hit files may also vary from site to site, additional dimensions are added to the options available to the data base administrator.

Data Compression

Compressing stored data is a technique to reduce the I/O required to service a query. The goal is to reduce the redundant or recurring information in the data base. For example, consider a social security number (SSN). If the SSN were stored as characters, at least nine and possibly eleven (there are two dashes) bytes would be required. If however the SSN's are stored as integers only 35 bits are required, a savings of nearly 3 to 1. Data compression can be applied in other ways. It is often possible to drastically reduce the amounts of storage used by compressing the data using statistical techniques. Essentially, we establish a compact encoding for the data to be compressed. In such coding schemes the most frequently occurring data values are represented by only a few bits. Statistical encoding schemes do suffer from three problems. First, values are encoded with variable length bit strings, potentially increasing the complexity of manipulating the data. Second, an auxiliary data structure is needed to decompress the data. Third, the natural ordering of the data may be lost.

If data can be maintained and manipulated in its compressed form, additional benefits accrue. In such a scheme, the user's query would be compressed also. All comparisons will be made between the compressed values, speeding up the comparisons. The data would be shipped over the network in its compressed form, which would increase the effective bandwidth of the network. By manipulating the data in its compressed form, overall effective throughput can be increased by a factor approaching the compression ratio.

Data compression is a well established technology, and we are not proposing to advance the state of the art in it. The experimental system must be designed to allow compression techniques to be employed throughout. We need to be able to run controlled experiments on the impact and effects of compression. It is also important that the techniques for and impact of data compression be disseminated to the WWMCCS community.

Clustering

Clustering refers to the technique of reducing I/O requirements by making the physical structure of the data base conform to user access patterns. For example, we could store all the data which satisfied a commonly occurring query in consecutive physical blocks. In many existing data base systems only a small percentage of the data brought in from secondary storage actually satisfies the user request. This increases the number of I/O requests needed to satisfy the query, degrading response time. The experimental system must be able to support a variety of clustering schemes. In particular it must allow a comprehensive internal data collection capability so that new clustering techniques may be developed, implemented, and analyzed.

Integrity

In a distributed data management system a single mechanism is responsible for the maintenance of the data base, although that mechanism may be implemented as multiple processes running on multiple hosts. All maintenance activity is coordinated in the same way. In contemporary environments data is captured in different ways, is subjected to varying degrees of scrutiny and authentication, and is inserted into disparate data management systems. It is not surprising that severe integrity (consistency) problems exist when data from several different systems is merged. A distributed data management system can help solve some of the integrity problems outlined above. However, a distributed system does introduce new problems. Because copies of data bases are being maintained at multiple sites, coordination of updates is essential. The major research need in this area is to provide effective synchronization mechanisms.

The experimental system should provide a framework for further research in integrity. As a longer term issue, the system can be used to investigate those aspects of integrity which are more closely related to the user's view of the world, i.e. the data capture and validation problems.

Required Attributes of the Experimental System

Introduction

Many system features are considered necessary to provide a flexible experimental tool. Some of the most important ones and a brief discussion of factors motivating them are presented here.

Transparency of Storage Structure and Data Location to Application Programs

Description. A user application program must not contain implicit or explicit assumptions about the structure of the underlying data management system (DMS). All interfaces between the DMS and programs must take take place via well-defined, appropriately-chosen primitives which do not make such assumptions necessary.

Motivation. Since the system is experimental in nature, it is expected that major changes in approach will be made frequently. The changes will occur both during preliminary design and during concept testing and experimentation. Because of the expense of designing and constructing suitable test programs and the desirability of uniform benchmarking during experimentation, it is highly desirable to be able to use a single set of test programs, essentially without modification, for all testing. Also, since the DMS and test programs are essentially independent, their design and construction can proceed independently without unnecessary delay caused by design changes. Another very important advantage to be gained by the independence of programs and the DMS is the possibility of implementing versions of the DMS on different host systems, each one being optimized internally for operation on its respective host, but presenting a uniform appearance to application programs.

Freedom of Structure Realizations

Description. A rigid structure must not be imposed on the data model for artificial reasons, e.g. to fit some computer manufacturer's de facto standard. Within the data model chosen, all possible flexibility will be retained to allow unanticipated changes to be made as easily as possible.

Motivation. The ideal structures to represent a data base are not obvious. Tradeoffs must be made between factors such as overhead for expressing structure, structure creation cost, updating cost, and retrieval time. If some of these factors are rigidly controlled by the data model or the cost of changing the structure of the experimental data bases, fruitful research paths may be bypassed.

Flexibility in Indexing of the Data Base

Description. When indexing of a data base on some variable is deemed necessary or desirable, independent indexing structures can be generated. These indexing structures need not be defined when the data base is created, and are visible to the user only by their effects on query response time. Their presence is solely an efficiency consideration, and is not required for correct operation.

Motivation. When executing query operations, an index structure can often reduce or eliminate the need for searching a data base. However, a good choice of variables to index depends heavily on the query patterns, and may in fact change on a short-term basis. If the index structure is part of the data base, a restructuring of the entire data base is usually necessary to change it. However, if the index structure is separate, it can be created and destroyed without affecting the data base. This capability opens new areas for experimentation, particularly in the area of resource utilization vs. response time tradeoff optimization.

<u>Simple and Modular Data Model and System</u>

<u>Description</u>. The data model must be conceptually simple and easy to experiment with. The DMS itself must be easily understood and easily modified.

<u>Motivation</u>. Because of the experimental support role the system is intended to play, the majority of time spent on the system should be in areas which actively contribute to the research goals. Time spent understanding complex aspects of the system, unless they bear directly on some experimental goal, can be considered wasted. Such wasted time must be minimized.

<u>Exploitation of Parallelism</u>

<u>Description</u>. Often, many of the operations required to perform a query can logically be performed simultaneously. The data model used must not limit or restrict the use of parallel evaluation more than absolutely necessary for integrity/synchronization purposes.

<u>Motivation</u>. In a network environment, the opportunities for optimization of query response by the exploitation of implicit parallelism are greatly increased. The unique possibilities opened by the availability of several computers, each with possibly unique capabilities, must not be prematurely or unnecessarily restricted. In order to experiment in this area, a non-restrictive theoretical model must be used. Restrictions can be imposed upon such a structure without difficulty, whereas if a more restrictive theoretical base were chosen, some experiments would be impossible or too contrived to give useful results.

Simple and Modular User Interface

Description. The system, as viewed by the user, must be simple enough for new users to learn with very little instruction and practice. Complex and/or non-intuitive features must be avoided as much as possible. The interface must be of modular design so it can easily be replaced by a different one.

Motivation. Since the system is not intended for production use, only those features necessary to carry out the desired experiments need be implemented. More importantly, the users of the system will generally be concerned primarily with the operations the system performs in getting results and not so much with the results themselves. In this context, learning effort and complex command entry represent wasted research time and hence should be minimized. Additionally, if end-user comment, such as from potential WIN users, is desired on some point, the familiarization time for a novice user must be reasonably short to minimize wasted personnel time. In the event that such comment suggests major changes to the user interface or suggests more extensive user testing, the interface must be easily modified or replaced to increase its usability.

Data Model Selection

Summary. Three data models are considered: hierarchical, plex (network) and relational. The major differences among the models are the kinds of structures which the user must define and utilize in addition to the actual data. The relational data model has been selected for the experimental system because of its greater flexibility and conceptual simplicity. An analysis and brief tutorial follows for each model.

Hierarchial model. The hierarchial model allows only simple, tree-like structures (Figure 1).

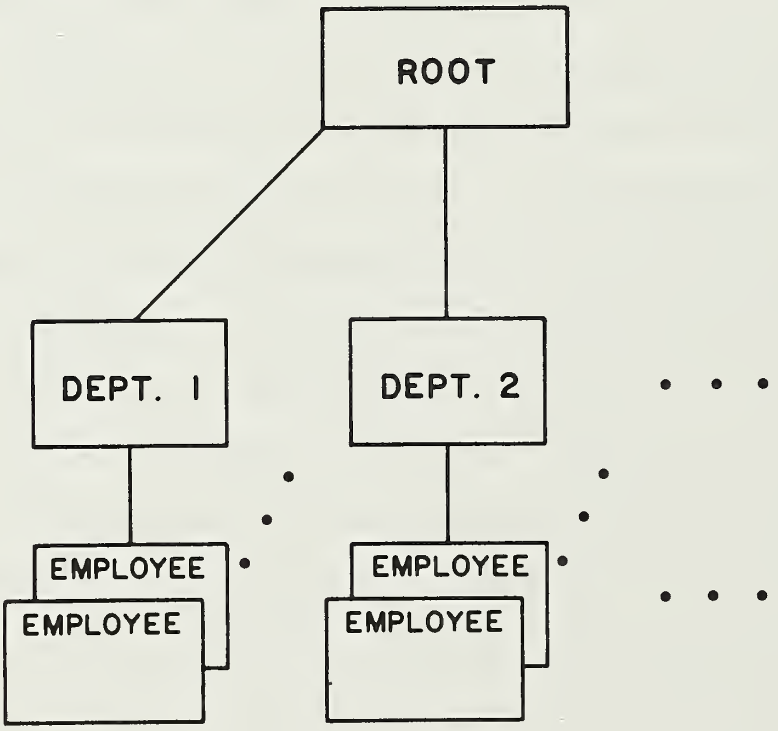

Figure 1

Hierarchial Structure

An important feature of this data model is that information is contained in the structure as well as in the data stored in that structure. A part of the tree cannot be separated and used independently without also transferring information about where that part was in the original tree. For example, in Figure 1, the information about the department where an employee works is contained in the structure of the hierarchy, i.e. the "owner" department for an "owned" employee. This data model has the chief advantage of being closely analogous to the physical storage representation of the data base. This can lead to improved processing efficiency. However, this feature can also be a major disadvantage. Suppose it is desired to access the data base in a different way, such as finding out whether any employees work in two departments. Because the department-to-employee relationship is manifest in the structure of the tree, it is not easy to extract the information in a different form, i.e. employee-to-department. Considerable work has gone into the translation of data bases, in order to be able to produce a new data base with, for example, an employee-to-department structure.

In a distributed environment, additional disadvantages are associated with the hierarchial model. Due to structural considerations, the "units of distribution" are complete subtrees. That is, it is not reasonable to distribute an arbitrary collection of data segments. If this were done with no restrictions, it might become necessary to "bounce" between two hosts in traversing the data base. The problems of information content in the structure continue to be bothersome. Experiments in restructuring and maintaining several copies of the same data in different physical structures would be difficult. Finally, it is potentially quite difficult to exploit the parallelism of a network.

Going into a hierarchial structure is a sequential process because the path taken conveys information. Thus, a hierarchial structure would not appear to be well-suited for the experimental system.

Plex Model. The plex model is an extension of the hierarchial model which allows a much wider variety of linkages between the data segments. Thus, hierarchies may be created as special cases. The CODASYL Data Base Task Group (DBTG) has presented one of the more complete plex models. Their model allows this additional structure to be expressed as SET membership. Such membership is manifest, conceptually at least, as chains of data segments (which DBTG refers to as record occurrences). Each record occurrence may be linked to a predecessor, a successor and to a so-called set-owner record. The other concept from the DBTG work which we shall discuss is that of an AREA. Basically, an AREA is a mechanism by which the data base administrator (DBA) can partition the data base into more manageable portions. The DBA can control the placement of the AREA on physical storage. The AREA may also be employed as a logical construct to be used for clustering, such as the grouping together of records to be used by a procedure. The CODASYL DBTG schema (data base structure definition) may be restricted so as to minimize linkages from one area into another.

A plex system would be potentially an attractive data model were it not for the lack of flexibility explicit structure definitions impose. Because the user is aware of these structures, they cannot be changed at different copies of, say, an AREA. Each copy must preserve the logical structure defined in the schema. Another practical limiting factor is the fundamental complexity of plex structures and their definition. Considerable manpower would have to be invested in schema analyzers etc. This manpower could be better utilized investigating questions which are truly distributed data base issues. Finally, there is the question of performing controllable experiments. In a plex model, the structure may induce inherent inefficiencies. For example, some queries simply cannot be answered efficiently due to the structure of the data base. It would be difficult to separate out the effects of the data base structure in an experiment done with such a system.

Relational model. The relational model was proposed as an attempt to improve the theoretical foundations of data base management and to provide a very simple user structure. In the short time since its initial suggestion, the relational model has generated tremendous interest within the academic and research communities. We have selected a relational data model because of its simplicity and flexibility.

A relation, R, may be thought of as a table whose columns are labeled. Since the columns (which are called _domains_) are always referred to by name, their order is unimportant. The number of domains is called the _degree of R_. The rows of the table (which are called _tuples_) have the following properties:

1. each tuple is distinct,
2. the ordering of the tuples is insignificant, and
3. the domain values in the tuple are atomic, e.g. character strings and integers but not structures or repeating groups.

Why has such a simple concept generated such interest? First, because the model is so simple, even the most computer-naive user can understand it. Second, because the relational model has no user-supplied structure, the system designer and data base administrator are free to provide any structure they wish "underneath" the user interface. The DBA is free to index, sort, or cluster a relation in any way he wishes. Further, such structures may be changed at will, with the only impact being on system response times. The user interface is fundamentally an associative view of the data. He supplies criteria and the system gives back the tuples meeting those criteria.

This is a good place to define the operators for the relational model, even though such definitions do not impact upon the selection of a data model. These definitions will be very brief, being sufficient only to understand the examples and terminology used throughout the sequel. Previous work on the relational model has led to the definition of two different sets of operators: the relational calculus and the relational algebra. In the calculus the user gives a first-order predicate calculus description of what he wants his output tuples to be like. In the algebra the user describes explicitly the operations necessary to achieve a desired output relation. We have chosen an algebra context for three major reasons:

1. The algebra deals with relations as objects rather than tuples. It is thus at a somewhat higher conceptual level.
2. The algebra is a close approximation to the way in which the system would actually be implemented.
3. The calculus requires additional transformations to determine what operations need to be performed to satisfy a query; the algebra is more explicit.

Since it can be shown that the two languages are comparable in their expressiveness, there is no loss of generality by our choice of the algebra for examples.

Only three operators will be used in subsequent examples, projection, restriction and join. Each takes one or more relations as operands and produces a relation as output. To aid in the definition, examples using the following relations will be presented.

SUPPLIER:

SPLR.NAME	SPLR.#	STATUS	CITY
Ajax	10	9	Tuscola
Acme	20	5	Arcola
Widget	30	6	Mattoon
Bomad	40	8	Urbana

PARTS:

PART.#	PART.NAME	COLOR	WEIGHT
1234	widget	blue	100
1235	widget	green	100
1236	bolt	blue	1
1237	bolt	green	1
1238	nut	purple	1
1239	nut	blue	1

SP:

SPLR.#	PART.#
10	1234
10	1235
10	1238
20	1239
20	1237
30	1234
30	1235
40	1236
40	1238

Projection takes as input a relation and an ordered domain list and produces a relation containing only the domains specified in the domain list. For example, to produce a list of the suppliers and their locations we write:

SUPPLIER [SPLR.NAME, CITY]

which produces the following relation:

SPLR.NAME	CITY
Ajax	Tuscola
Acme	Arcola
Widget	Mattoon
Bomad	Urbana

The domain list is enclosed in square brackets immediately after the relation. Projection may also be used to simply re-order the domains of a relation by including all of them in the projection domain list.

Restriction takes as inputs a relation and a Boolean expression of the domains of that relation. It produces a relation with the same domains as the original input relation. The tuples of the output relation are exactly the tuples of the input relation which satisfy the input Boolean expression. Strictly speaking, restriction is redundant, since its effects can be achieved through combinations of the other operators. However, it represents an easily understood and concise way of stating a commonly used operation. To find the suppliers whose status was greater than 6 we would write:

(SUPPLIER where STATUS > 6)

which would produce the following relation:

SPLR.NAME	SPLR.#	STATUS	CITY
Ajax	10	9	Tuscola
Bomad	40	8	Urbana

Restriction may be combined with projection, both by projection on the input relation or on the output. For example, to find the supplier names and status of suppliers whose status is greater than 6 we write:

(SUPPLIER where STATUS > 6) [SPLR.NAME, STATUS]

or

(SUPPLIER [SPLR.NAME, STATUS] where STATUS > 6)

Either one produces the following output relation:

SPLR.NAME	STATUS
Ajax	9
Bomad	8

As was shown in this last example, many alternative statements exist to do the same thing. This fact allows a variety of optimization techniques to be applied, as will be discussed later.

The simplest type of join takes as input two relations R1 and R2 that have a common domain D. It produces a new relation whose tuples are formed by concatenating a tuple from R1 to one from R2 with the same value for D. For notational convenience we will assume that D is the last domain of R1 and the first domain of R2. To find the supplier numbers of suppliers who supply green parts we write:

```
((PARTS where COLOR = green)[PART.#] * SP[PART.#,SPLR.#])[SPLR.#]
```

The join is denoted by the *. This could also have been written:

```
((PARTS [PART.#,COLOR] * {green})[PART.#] * SP[PART.#,SPLR.#])[SPLR.#]
```

Thus, join may be substituted for certain cases of restrict. Both these examples produce this output relation

SPLR.#
10
20
30

The following example may help to show what join does. We will go through join on a tuple by tuple basis.

```
SP[SPLR.#,PART.#] * PARTS [PART.#,PART.NAME,COLOR,WEIGHT]
```

taking the first tuple from SP, <10, 1234>, we find the tuple in PARTS that has the same value for PART.#, <1234, widget, blue, 100>. The concatenated tuple <10, 1234, widget, blue, 100> is now output, it is in the output join relation. This process is repeated for each tuple in SP. Of particular interest is the SP tuple <30, 1234>. Semantically this means that both supplier 10 (Ajax) and supplier 30 (Widget) supply part number 1234, the blue, 100 pound widget. Thus, both the following tuples are in the join:

<10, 1234, widget, blue, 100>

and

<30, 1234, widget, blue, 100>

This example eventually produces the following output relation:

SPLR.#	PART.#	PART.NAME	COLOR	WEIGHT
10	1234	widget	blue	100
10	1235	widget	green	100
10	1238	nut	purple	1
20	1239	nut	blue	1
20	1237	bolt	green	1
30	1234	widget	blue	100
30	1235	widget	green	100
40	1236	bolt	blue	1
40	1238	nut	purple	1

The structure of the three relations in this example results from a process called normalization. Discussion of normalization is beyond the scope of this document. However, its practical significance is that the large conceptual relation that describes parts and their suppliers must be broken down into SUPPLIER, PARTS and SP. This breakup avoids certain types of problems associated with updates and deletions. Unfortunately, this breaking up of the conceptual relation means that the user must remember much more obscuring detail. For example to find the name of the supplier of some part, two joins are necessary. In an attempt to alleviate the problem the notion of a view has been introduced.

A view is the relation that the user deals with. A view is made from operations applied to the basic relations of the system. For example, suppose we have a user who only works with widgets. There is no need for him to be aware of the existence of other types of parts. Further suppose that all he does is write orders to the widget suppliers, Ajax and Widget. Our user has no need for the supplier number and should not be burdened with projections to omit it. This user would have a view defined by the following expression:

```
     VIEW1 = (SUPPLIER * SP * (PARTS where PART.NAME = widget))
                    [SPLR.NAME, STATUS, CITY, PART.#, PART.NAME, COLOR, WEIGHT]
```

(The projections needed for the joins are omitted for clarity.)

To this user the data base looks like the single relation VIEW1:

VIEW1:

SPLR.NAME	STATUS	CITY	PART.#	PART.NAME	COLOR	WEIGHT
Ajax	9	Tuscola	1234	widget	blue	100
Widget	6	Mattoon	1234	widget	blue	100
Ajax	9	Tuscola	1235	widget	green	100
Widget	6	Mattoon	1235	widget	green	100

All of his queries may be directed towards VIEW1, rather than the underlying relations.

In a distributed DMS a relation may actually be made up from pieces stored at several network hosts. We shall call this concept inverse views. An inverse view would be useful if most of the use of the inverse view is local, but occasionally the entire relation is needed. For example, the FORSTAT data bases now form an inverse view. Each command operates on a portion of a large conceptual data base. This conceptual data base is the FORSTAT for every unit in every command.

A distributed data base requires a great deal of flexibility from the underlying data model. Duplicate copies of a portion of the data base should be free to be organized in any way. This will allow for experimentation with the effects of a heterogeneous data base even when done on a homogeneous network. A given relation may be replicated at different hosts. If each copy has a structure which is optimized for particular types of retrieval, the distributed DMS will be free to select the most suitable copy. A user request which requires access to multiple relations may be able to exploit the parallelism of the network by utilizing copies of relations which are stored on different hosts. Finally, by choosing a simple model like the relational model, controlled experiments can be performed. These experiments will not be clouded by the effects of the data structure.

Machine Selection

Summary. Multics at MIT will be used as the primary software development site. Multi-site experiments can be supported using additional Multics sites on the ARPA Network. The decision to use the ARPA Network and Multics is based primarily of the availability of those resources, significant local expertise and the clearly superior support Multics offers to the software developer.

Criteria. The purpose of the experimental distributed DMS is to support experimentation. Consequently, the target computer system selection must be based on the effectiveness of the target system as a program development tool. This factor outweighs conventional criteria such as cost and locality, since programmer time is limited and must be used effectively. Other factors must be considered including:

1. cost,
2. accessability,
3. short-term reliability and availability,

4. availability for the duration of the research period,
5. response time for interactive use,
6. local expertise,
7. possibility of transporting the distributed DMS to other systems,
8. possibility of use of pieces of an existing DMS,
9. availability of suitable language(s) on the system,
10. availability of suitable network access, and
11. availability of at least two systems on the network for testing purposes.

A detailed comparison of the available systems would be expensive and would probably be of little value, since weightings for the various factors would be largely subjective and hence could be chosen to favor any desired system. A subjective choice, provided it is reasonable, might as well be made and identified as such.

The designers of any system draw on their own past experience when making otherwise non-obvious choices. Some conclusions based on past experiences of this design team will be described here so the reader can better understand the background of choices described later.

The operating system support must be extensive. The construction of large, complex programs almost invariably uncovers obscure operating system problems. These problems may be circumventable, but often are not. Most manufacturer-provided operating systems or contractor-written support software cannot be fixed in a matter of months, much less the hours that users would prefer.

The physical operation of the facility must be professional and provide adequate services. When a user who is using a system from a terminal requests that an operator mount an offline storage volume (e.g., tape or disk), every second of delay costs the user time, money, and inconvenience. Some systems require literally hours, for example, to retrieve a file from an archive tape. This is unacceptable. Operator mistakes are also unacceptable. Proper operator interface design and good operator training can essentially eliminate them. Operating hours must be reasonable.

Programmers should not need to be overly concerned with program size and file storage space. While programmers should clearly be concerned with storage efficiency and wasted file storage space, these must not be allowed to become major concerns. The authors know of cases where man-months have been spent setting up overlay structures for large programs to pare as little as 30% from the memory requirement. Similar anecdotes probably exist for every non-virtual-memory machine in existence. This group considers a non-limiting memory size an absolute necessity to avoid totally fruitless, but time-consuming effort. Similarly, file-storage-limited machines can cause tremendous amounts of effort to be wasted in shuffling files on and off the system. In one case in the authors' experience, over 20% of all programmer time spent over a several-month period by a team of five programmers was spent in file maintenance due to problems caused by limited file space.

Appropriate high-level programming languages and debugging and measurement tools must be provided. The disadvantages of assembly languages are well-documented in the literature. The originally perceived advantages, namely reduced size and speed of the resulting code, have been consistently disproven for moderate and large programs. For debugging purposes, an interactive symbolic debugging tool is essential.

Learning a machine in sufficient detail to use memory dumps is costly and unnecessary.

Alternatives. Many alternatives are available for the choice of a host system for the distributed DMS. The availability of the ARPA network provides a variety of machines and operating systems to choose from. Other possibilities are local facilities at the University of Illinois and GCOS time obtained via the PWIN. Local facilities at the University of Illinois are not networked and thus cannot support networking experiments. Use of the PWIN will require unclassified operation. The amount of time available for this mode of operation is very limited and would severely slow the research program. The ARPA network is available and adequate to the research needs.

The CAC has over four years of experience using virtually all of the hosts on the ARPA network. Furthermore, we already make extensive use of the three major service hosts: UCLA's 360/91, BBN's TENEX and MIT's Multics. (A fourth host on which we have considerable experience, UCSD's B6700, left the net this summer.) All three sites provide a production-oriented site operation adequate for our purposes. Of the three, Multics is the only one which meets all other criteria. UCLA does not provide a virtual memory system and TENEX does not provide an appropriate high-level language. Furthermore, costs at Multics are favorable and our past experience has shown that the software development tools available in Multics are advanced well beyond its nearest competitor. There are currently three Multics machines on the ARPA network. The two Multics sites other than MIT do not have an appropriate production orientation for continued software dvelopment, but they are adequate for multi-site experiments. In view of the commanding advantages of Multics for this particular research activity, it is clearly the machine of choice.

Application to PWIN/WIN

This section discusses how the choice of a relational data model to be implemented on Multics relates to the PWIN/WIN environment. The technical grounds for these choices are clear. The relational model captures the essence of the other competing data models while allowing unparalleled freedom in experimentation. Multics represents perhaps the best system in existence for support of experimental software development.

Obviously, it would be an error to use a data model which, although elegant, could not be related to the realities of the PWIN/WIN environment. The transfer to the WWMCCS environment of results obtained from a system based upon a relational model may not be obvious. The relational model, in some ways, represents a "least common denominator" for data base systems because all the data associated with an entity is explicitly specified. A tuple is an atomic unit with no need for any context. Thus, controlled experiments can be performed on the effects of different organizations. These experiments can lead to better understanding of the general principles underlying any data management system.

The Data Translation Project at the University of Michigan has demonstrated that typical WWMCCS data bases may be translated into and out of relational form. Indeed, a central concept in their data translation system is the use of a relational form as a common intermediate structure when translating between two dissimilar data base structures. Thus, the experimental system could use a typical or representative

WWMCCS data base translated into relational form. Experiments performed upon the relational form might suggest reorganizations of the original data base to improve efficiency. In a distributed environment, we plan, as a longer range project, to investigate the impact of having different structures for the copies of a data base. In particular, we want to assess the effect of limitations on the structure resulting from restrictions on the schema to sub-schema mapping. This knowledge can be directly applied in the design of functional specifications for an operational distributed data base system.

The choice of Multics is also relevant to the WWMCCS community. Multics hardware is quite similar to the WWMCCS standard hardware, particularly the peripherals. It may be possible to infer timings on GCOS machines from experiments performed on Multics.

Experimental System Design

System Overview

This section discusses the design of the experimental distributed data management system. Because this is a first cut at the design, we have tried to focus mainly upon the underlying issues. For the purposes of this document the system has been broken down into three components: the network data manager, the local data manager, and the network data base administrator support functions. In the actual implementation of the system, it is likely that the distinction between these components will become blurred.

To illustrate the relationship among the three components, we will begin by sketching the actions which occur in satisfying a query. In the rest of this section the three system components will be discussed in more detail.

The processing of the user query starts by parsing it into some internal form. This command language processing is best included in the discussions on the network data manager. The query (now in an internal form) then passes through two major stages.

1. Global optimization. The query is examined to determine the best way to satisfy it. This process may involve a transformation of the query into a more efficient, but equivalent form (e.g., change the order in which the operators are evaluated). Included in the optimization is the selection of hosts on which to perform the required operations.

2. Query evaluation. This may be further broken down into two steps: setup and execution. In the setup phase, the processes which will perform the required operations at the various

hosts are initialized. These include local data managers which will actually retrieve data from stored relations, and the service functions of other network data managers. The network data manager service functions are the modules which perform operations like join on the outputs of local data managers. In the execution phase, the originating network data manager issues commands to the local and network data managers working on the query. As tuples which satisfy the query are processed, they are sent to the required destination.

Occasionally a host may fail. The network data managers on surviving hosts must take the actions needed to switch over to backup copies of data bases. If, as is likely, the failed host was in the middle of processing queries, several possibilities exist. The network data managers who originated the queries may be able to salvage the partial results produced before the host failed. A backup copy can then be used to complete the query. Alternatively, the originating network data manager may decide to discard the intermediate results and restart the query at the optimization step. This new optimization would, of course, be performed based on the network configuration after the host failure.

Throughout the system, statistics are being collected by the various network and local data managers for use by the network data base administrator. The network DBA may elect to move relations, add or delete indices, etc. based upon these statistics.

Network Data Manager

Overview. The functions of the network data manager are:

1. to accept user requests and process the commands into an internal form,

2. to perform a global optimization on the user's query by re-ordering operations and determining where (at which hosts) the operations needed to satisfy a request will occur,

3. to initiate and monitor these operations, and

4. to perform relational operations in response to requests from other hosts.

The last function is not, strictly speaking, part of the network data manager, but this is the natural place to include the discussion. In the example which immediately preceded this section, a typical query was followed through these various functions. In this section each function will be discussed in somewhat more detail. The emphasis throughout this discussion on the experimental system design has been on the most basic considerations. It is somewhat premature to offer a detailed design of the system.

Command language processing. The user language must be as simple as possible while still providing the necessary power of expression. This is in keeping with the philosophy of simplicity described earlier. The language will be a command and query language with interpretive execution.

In order to execute a query, the form of the query must be checked to verify its syntactic and semantic correctness, and then transformed into a form suitable for computer interpretation. This process is called parsing and translation, and resembles the first steps performed by a high-level-language compiler.

The computer-usable form of the query, i.e., the query after parsing and translation, will be an n-ary tree (probably binary). A tree structure has the advantage of clearly showing the flow of data necessary to satisfy a request. Since the limited network bandwidth can be expected to be an important consideration in optimizing the retrieval strategy, a tree is good internal form. A typical tree is shown in Figure 2. The nodes labeled REL are basic relations. A basic relation is simply a relation that is stored on some storage medium or available via a network. The nodes labeled OP are operators. An operator performs some operation (e.g., a Boolean operation, a selection process, a sort, etc.) on one or more relations. The operators need not know whether the relations they operate upon are primitive relations or the results from previously applied (lower in the tree) operators.

<u>Global optimizer</u>. It is often possible to greatly reduce the effort required to fulfill a query by rearranging the order in which operations are performed. For example, consider the tree in Figure 3. This tree might represent the query (in English):

"Show me the amount of income tax paid by all managers making over $30,000.00 per year."

If REL_1 relates employee number to income and job position, and REL_2 relates employee number to income tax paid, the join operator in Figure 3 produces a composite relation which relates employee number to income, job position, and income tax. The restriction operator then removes those employees who are not managers making over $30,000/year, leaving the result desired. In Figure 4 the restriction has been moved. In this tree, the restriction produces only those employees who are managers making over $30,000. The join then produces the desired result. The restriction operator was probably only a little cheaper to perform in this case because it was handling relations with fewer fields (domains). However, the join operation only had to process those employees from REL_1 which met the criteria, which will usually be a much smaller number

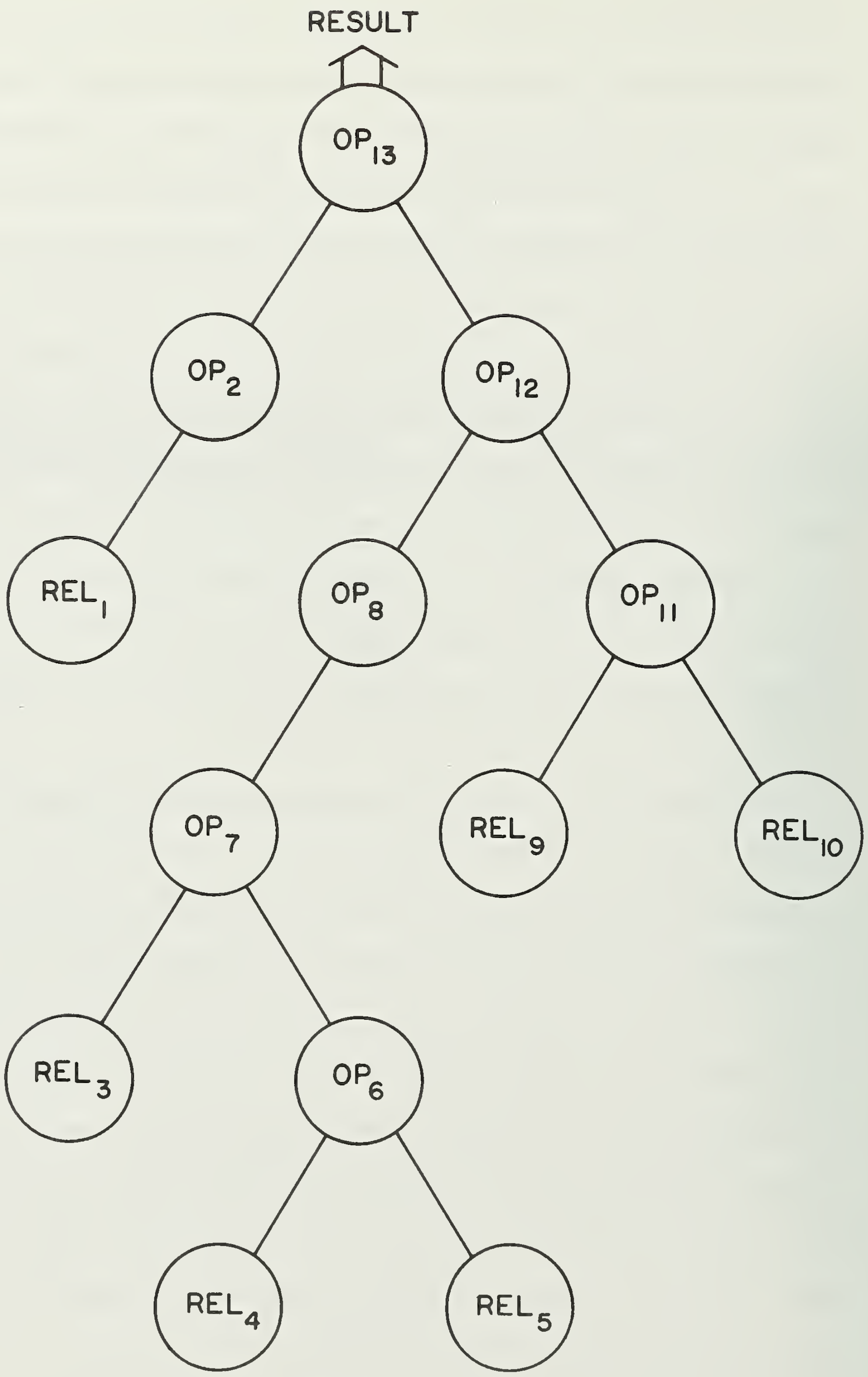

Figure 2

Query expressed as a tree

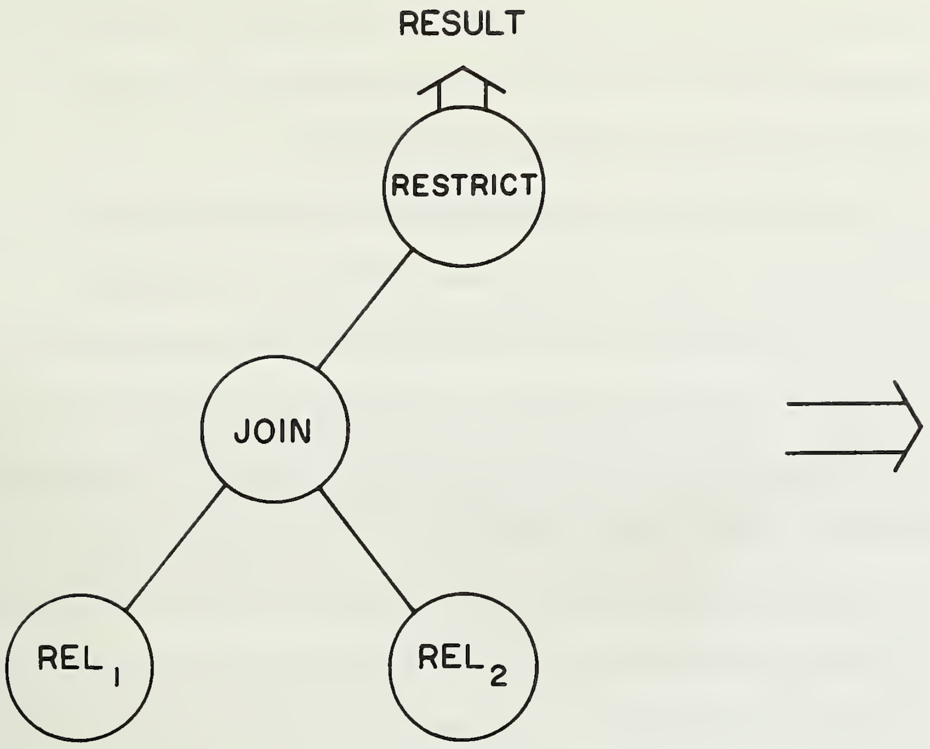

Figure 3

Query tree before optimization

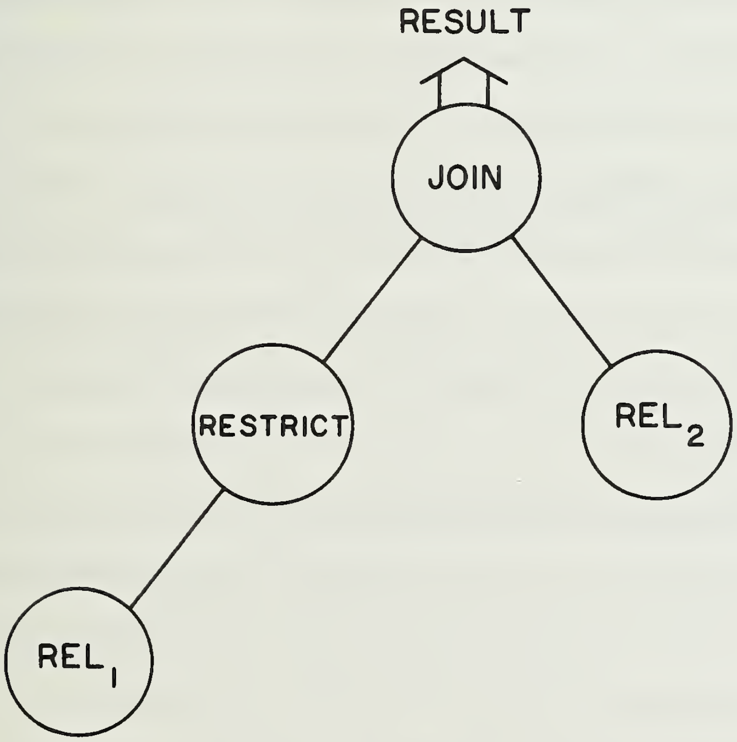

Figure 4

Query tree after optimization

than the total employee relation. Hence, the latter tree will probably be at least as efficient as the first and usually more efficient unless all employees are managers making over $30,000.

The above example demonstrates one small aspect of high-level query optimization. There are several classes of optimization that can be used, and optimization can be performed with many different objectives, e.g., minimizing cost, time, or resource utilization (such as temporary storage), to fulfill the query. A few of the techniques for optimization that can be used follow.

1. Query tuning. This generally refers to using knowledge or guesses about the nature of the data base to rearrange the order of operations. In the above example, if it were known that all managers made under $30,000, the query could be filled (vacuously) by the optimizer.
2. Rearrangement of operators. The example above is a simple case of operator rearrangement. Frequently, the improvement is much more dramatic than in this simple example.
3. Optimizing according to availability of indices. By rearranging the tree, it is possible that evaluation can take advantage of sort order or indices on one or more relations. Either can be used to eliminate the need for searching an entire relation.
4. Optimizing by locations of relations. This is discussed in the next section. As a quick example, it might be possible to minimize network transmission by using two relations located at a single host.
5. Optimizing for maximum parallelism. This may involve performing part of the query at different hosts or arranging the

order of evaluation so that the expression tree is as "broad" as possible but no "deeper" than absolutely necessary. ("Broad" refers to the number of operators at each level in the tree, "deep" refers to the number of levels.) The former case is discussed in the next section.

The optimization problem can be considered to be that of minimizing the value of a cost function which takes all pertinent factors into account. Such a function would be developed by the Modeling effort.

Query evaluation. As described earlier, the query entered by a user is parsed and transformed into a tree (Figure 2). The query would be evaluated by traversing the tree and evaluating each node as it was visited. The order of visiting must be chosen so that the arguments to an operator are available when the operator is evaluated. (One possible order of evaluation of the tree in Figure 2 is reflected in the node numbering shown.) One problem with this simple method is that any potential parallelism of evaluation cannot be exploited. Hence, we must use a more sophisticated evaluation method if we are to give prompt responses to queries. The following description is intended only to capture the flavor of the proposed technique. It has been modified and simplified for ease of presentation.

Part of the function performed by the tree optimization step is to decide where (i.e., at which host in the network) the parts of a query can be most efficiently evaluated. In general, a tree may be broken apart before evaluation, and spread through the network. To simplify the job of the query evaluator, all nodes of a tree are considered to be relations. This is true of actual relations stored on an external stroage medium, intermediate results of operators, and relations being computed at other locations and returned via the network. Only the routines used to physically access the relations may have to distinguish between local, remote, and intermediate relations.

In order to take advantage of parallelism and provide dynamic control over the size of intermediate files, the tree will not be evaluated serially, as described earlier. Instead, it will be evaluated, as much as possible, from the bottom to the top, in parallel. This allows adjustment of evaluation parameters (such as intermediate storage) to a greater degree possible than for serial evaluation and also permits any possible parallelism to be exploited.

In order to help exploit parallelism and reduce intermediate storage requirements, a technique referred to as "streaming" or "pipe lining" can often be used. This scheme consists of implementing operators which do not require entire relations to be present as inputs before producing an output. The data flows into such an operator in a "stream" and flows out in a "stream." For example, projection and restriction can operate on a streamed basis. In general, join and sorts must wait until all the needed tuples are avaialble. This technique, combined with the parallel evaluation technique above, produces the _first_ result tuple from a query as soon as possible, and produces further tuples as they are available. A normal evaluation technique, using intermediate storage ("hit files"), typically completes each operation before starting operations waiting for its results. The nature of the operators that can be streamed and the mechanics of the streaming operation itself are under study.

Service and utility functions. In the system described, there is no necessity that all the operations needed to fulfill a request be performed at the originating (user) site. Indeed, substantial inefficiencies could occur from transmitting large, intermediate relations over the network. So, a mechanism must be provided to perform service requests. For example, suppose a user on host A issues a request which can best be fulfilled by doing a join on two relations stored at host B. The network data manager at host A must inform the network data manager at host B of this decision. The server (host B) data managers cooperate to retrieve the tuples and send the results to the appropriate destination. Notice that the destination of the output might even be some third host. Considerable effort must go into the design of protocols to accomplish the desired communication. The user (host A) and server (host B) network data managers must also cooperate on the choice of the particular method of performing the desired operation. For example, the join should take advantage of any indices or sorts which can be used.

Many of the support functions required for the operation of the distributed DMS are not specifically DMS functions. This includes communication packages, file dump/restore capability, sort packages, and others. Clearly, the system used should provide as much of this service as possible. Some other DMS support functions are not used during normal operation of the DMS, but are part of the DMS. These include file and index integrity checking routines to aid in recovery and debugging, crash recovery programs, report generating programs, system operation monitoring routines, and others. This type of support must be designed and built, and may require a substantial amount of effort.

Local Data Manager

Introduction. The functions of the local data manager are:

1. to retrieve tuples from a basic relation,

2. to insert tuples into a basic relation, and

3. to gather statistics for use by the network data base administrator.

(By basic relation we mean one which is not derived from any other relations.) As was briefly mentioned in the system overview, the local data manager resident at a host operates only on the data local to that host. The precise division of labor between the network and local data managers is likely to be smeared in an implementation. For example, considerable efficiency can be gained by the local data manager's being aware of what use is intended for the retrieved tuples. In this section we will discuss each of these functions.

Retrieval. For retrieval, a network data manager sends to the local data manager a relation name R, a selection function F, which is either true or false for each tuple in R, and a sort criterion for the output tuples. The local data manager must extract all the tuples from R for which the selection function is true and supply them to the network data manager in the appropriate sorted order. Both the selection function and the sort criterion may be null.

If the local data manager were to perform these tasks in a brute force manner, its performance would be totally unacceptable. Rather, the local data manager must be able to employ aids such as indices, clustering, and expression optimization to make the data retrieval as efficient as possible. To discuss these concepts more completely, it is necessary to have some additional definitions.

Each domain of a relation may have an index defined for it. This index is an additional data structure which relates the values of that domain to (sets of) tuples (possibly) containing those values. A domain is completely inverted if, given a value, we can always determine from the index exactly which tuples contain the value. A domain is partially inverted if, given a value, we can determine from the index a set of tuples guaranteed to include all the tuples containing the value, but possibly including some others.

A relation may be clustered on a domain (or collection of domains). Clustering can improve the efficiency of accessing the relation because we can improve the hit rate (i.e., a greater percentage of the tuples accessed will satisfy the retrieval specifications). Thus, fewer I/O operations are necessary to retrieve all qualified tuples. Clustering may be combined with indexing.

The work necessary to reply to a retrieval request on relation R, satisfying qualification function F and sorted by sort criterion S is:

W(R,F,S) = work to process indices + work to retrieve tuples
+ work to qualify retrieved tuples
+ work to sort qualified tuples

What we seek is to minimize the total work to reply to the retrieval request. However, since the terms are related to each other, we cannot simply minimize each term. In the following paragraphs we will discuss each term and some of the interdependencies.

The starting point in optimizing local retrieval is the qualification function F. Let us consider functions which are Boolean expressions of domains and constants, such as the following:

F1: AGE < 25 and SALARY < $5000

F2: PART COLOR ≠ GREEN and SUPPLIER ≠ JONES

F3: SALARY > (2*STD DEVIATION(SALARY) + AVG(SALARY))

We can minimize the work to retrieve tuples by using indices where possible. For example, with F1 as the qualification function, an index (complete or partial) for either AGE or SALARY could be profitably used to cut down the number of tuples to be retrieved. We could process the index to determine in advance which tuples could not possibly satisfy F1. If both AGE and SALARY were indexed, we could perform an intersection on the two indices and retrieve even fewer tuples. This does increase the work expended in index processing, and also may decrease the work in qualification. (If both AGE and SALARY are completely inverted, for example, every tuple retrieved satisfies the qualification expression.) Thus, indices should be used when they substantially reduce the number of tuples to be retrieved. In the second example it is probably not advantageous to use indices. Suppose both PART COLOR and SUPPLIER are completely inverted. We might spend considerable time intersecting indices to exclude only a few tuples. This can result with any comparison operator and is not simply an effect of the $\neq$. In one intelligence document data base that the authors are familiar with, over 80% of the documents had SOURCE COUNTRY = USSR. The system designers simply prohibited the use of that comparison in retrieval qualification functions.

An index for SALARY would be of little value if F3 were the qualification function. There, the values of interest are not a constant but are derived from operations on the domains of the relation. In general, expressions where two domains are compared are not amenable to index processing. If queries like F3 are frequently processed, we may wish to calculate the AVG and STD DEVIATION and remember them for future use.

These examples are intended to illustrate the non-trivial nature of index processing. We are continuing to study the subject in an effort to determine the general principles involved. These particular examples are all relatively clear cut; in a practical system, decisions will often have to be made in far less obvious circumstances.

There is also a great deal of work to be done on effective index structures. Ideally, one would like the index to yield the desired tuple identifiers after a minimum of processing. In practice, the structure which is optimal for some queries may prove to be totally unsuited to others. Very little theoretical work has appeared on the subject. We plan to consider a wide variety of structures. By modularizing the index functions, we can experiment with different alternatives with a minimum of effort.

The actual mechanics of tuple retrieval are heavily dependent upon the physical realization of the relation. After index processing we have a collection of tuple identifiers (or identifiers of blocks of tuples from a partially inverted domain index). Conceptually we use these identifiers as pointers into an array of tuples. Practically, there may be transformations applied to achieve the identifier-to-tuple mapping. At the tuple level, considerable freedom exists as to the physical realization of the domains. In one system (RDMS on Multics) which we have investigated, tuples are realized as a set of pointers into sorted arrays containing all the values for each domain. Such a technique has the advantage of having each tuple in a relation being a fixed length. Comparison operations may be performed on these pointers with the same effects as on the actual values. The disadvantage of this particular technique is that such indirection may cause additional I/O operations to be performed to actually retrieve the tuples. Another very attractive scheme would be to utilize a sophisticated data compression technique to reduce the size of the relation. The compression may be viewed as a form of indirection, since the values may be stored in a decompression table. The brute force methods may prove to eventually be an acceptable alternative. Here, a tuple is stored as a self-defining collection of domains. No indirection or compression is used.

As briefly mentioned above, the qualification of retrieved tuples may be profitably altered if we are able to use indices. Certain sub-expressions can be replaced with constant true or false values. These constants should be propagated as high as possible in the evaluation tree. For example, consider the following retrieval specification:

F4: AGE = 25 and SALARY > $20,000

If AGE is completely inverted, the qualification expression should become:

F4': SALARY > $20,000

This occurs because we can use the AGE index to guarantee that every tuple retrieved has AGE = 25. The exact order of evaluation should be optimized by taking into account known (or observed) characteristics of the retrieved tuples. A tuple should be qualified (or disqualified) as soon as possible. Ideally, we should use the probability of a condition being true for a tuple to reorder the evaluation. The theory here exists, but the practical difficulties of getting the required data obviate the effectiveness of the more elegant techniques.

After the tuples have been qualified, they may need to be sorted. The literature on sorting is very rich. There are several algorithms which would be suitable. If we are clever, however, it may be possible to avoid the actual sorting of the tuples. If the relation is already sorted on the appropriate domains, all we have to do is maintain that ordering through the subsequent processing of the tuples. This seemingly remote possibility may actually occur fairly often, (e.g., when previously normalized relations are joined back together). There, we would like both relations sorted on the joining domain so that a streamed join would be possible.

Insertion. Conceptually, insertion of a tuple into a relation is a trivial operation. The new tuple is simply appended to the relation. This conceptual simplicity is one of the major attractions of the relational model. In practice, it may be desirable to increase the complexity of insertions in order to speed up retrievals. Devices such as indices and sorts can greatly improve retrieval performance. They complicate insertion by requiring more objects to be altered to complete the operation. Since an insertion requires exclusive use of the relations being inserted into, including indices, the insertions may seriously restrict the system's retrieval capacity.

In a distributed environment the problems are heightened. All the copies of a relation must be updated. The different copies are likely to be organized in different ways. Views or inverse views may be utilized. Potentially the most vexatious problem is synchronization. Consider the following situation. There are two relations R1 and R2 located at different hosts. Each relation has a queue of user requests pending for it. We have just received a user request which requires insertion of a tuple in both R1 and R2. (This could come about, for example, if R1 and R2 were normalized relations which resulted from a single relation.) We must wait until both R1 and R2 are free. We cannot insert the tuple in R1 at one time, then later insert the tuple into R2. If this were done, operations like join would not work correctly, or worse, would give erroneous results. This synchronization scheme would leave us waiting for the last relation needed to come free. Meanwhile the previously locked relations would be unavailable for use.

Another technique for minimizing this problem is to have a reader copy of each relation and a writer copy, and periodically switch roles, doing a batch update on the former reader copy before making it the new writer copy. The problem with this second scheme is that we cannot see recent updates. In an on-line data base system, these may be precisely the ones in which we are most interested. Any attempt to read from the writer copy will return us to the problems of the first scheme.

The problem of synchronization in a distributed data base is, thus, not at all easy to solve. It is an area of continuing research. These two examples were intended merely to point out some of the difficulties.

Beyond the mechanical aspects of insertion lie the larger issues of integrity and consistency. Insertion is the most logical time to focus our integrity interests. A given tuple may be processed and verified before it is inserted. Certain fundamental integrity constraints may be simply considered in a relational model. These fundamental constraints relate to non-duplication of values in distinguished domains called primary keys, and to the preservation of the uniqueness of key domain to non-key domain mappings. Other integrity constraints can often be expressed as queries which must always (or never) be satisfied by an inserted tuple. Thus, if each insertion is processed against these integrity constraints, we can guarantee that they are met for the entire data base at all times. The local data manager may not always be the best place to include these broader integrity checks. For example, they may span several relations.

Measurement. The local data manager is the logical place to collect many of the statistics which will be used by the network data base administrator. These statistics would be the input for decisions on data base restructuring, index creation or deletion, clustering, allocation of relations to hosts, etc. All of these are basically static decisions. The network DBA makes or reviews his decisions at relatively infrequent times. The kinds of data needed for such decisions include:

1. summaries of the tuples returned for the retrieval requests received,
2. summaries of the types of qualification expressions used,
3. summaries of the output sort criteria used, and
4. measurements of the work expended in index processing.

This list is by no means exhaustive. As the system design is refined, the exact nature and characteristics of the necessary system measurements in support of the network DBA will become clearer.

Another class of measurements are of a more dynamic character. These are used as input for the various strategy decisions made for each query by both the network and local data managers. In the preceding parts of this section and in the discussion of the network data manager, we have shown how some of these measurements may be used. These dynamic measurements include:

1. responsiveness of the host,

2. probabilities of particular logical qualification expressions being satisfied,
3. limited retention of recent sort information, and
4. retention of derived values such as average values, counts, standard deviation, etc.

Again, as the system design is refined, the exact character of these measurements will become clearer.

Network DBA Support Functions

Introduction. This section is concerned with those functions which the network DBA performs to keep the distributed DMS running efficiently and smoothly. On existing single site systems he takes two classes of actions to reduce costs and improve response times. The first class is best typified by the reorganization of files on disk to maximize channel utilization. Any alternative will require the same amount of storage, and the same amount of I/O will be used by queries. However, response times will be improved because of the superior overlap of I/O and CPU.

The second class involves trading off the costs of two or more computer resources to optimize system efficiency. Such is the case when the DBA decides to add an index, to reduce the cost and time required for a class of queries. However, this is at the expense of index generation, and increased costs for updates storage. This type of decision must also be based on a very good understanding of current and anticipated user needs.

This problem is greatly complicated in a distributed environment where users with divergent interests are accessing the data base from a large geographic area. The network DBA would have to understand the time-varying needs of the user communities and the total load placed on the system by the interaction of these needs. As this is an formidable task, the system itself must provide sufficient performance statistics for the DBA to control the system's behavior. The first step is to determine those parameters which strongly influence system performance.

Much of the above description is equally applicable to a single site DBA. Indeed, a goal of a distributed DMS is transparency. The users need not be aware of the existence of the network or of the fact that not all the files are available locally. The major added problem for the network DBA is that remote files can only be accessed by relatively low bandwidth channels. The effective network bandwidth of 30 kbit/sec is a factor of over 200 lower than the transfer rates of .8 to 1. Megabyte/sec from advanced technology disk systems (e.g. HIS DSS 190, IBM 3330). Thus, the network DBA must carefully consider the allocation of files to the network hosts. The network DBA does have the flexibility added by the existence of multiple copies of files. As long as the information content of the copies remains the same, each copy may be optimized for a particular class of operation. The reliability advantages of multiple copies would still exist, and conflicting user requirements could be met.

In the remainder of this section we will discuss some of the functions the network DBA performs and the considerations that go into his decisions.

Definition of relations. When a new relation is defined, its domains, access control, and integrity constraints must be specified. As these decisions are based on real world knowledge, the DMS can do little to help the DBA. However, the DBA must also specify where to locate the relation, how many backups it should have, and their placement. There are three factors which influence these decisions.

First, the political constraints must be considered. A relation may be critical to a command's mission. This can force the relation to be stored locally. Even if communication with the rest of the network is broken, the command will be able to maintain its operations. Security considerations might also constrain which hosts could store a relation.

Second, the DBA must consider the impact of the relation on the performance of the various hosts. To assess this he needs to know the loads processed by the hosts and the amount this relation will impact load. Techniques for measuring system resources and the impact which another job would have on resource utilization are not well understood. Until such tools are developed, the DBA can make little use of this parameter.

Finally, the impact on network traffic must be determined. Other relations will often be involved when evaluating a query. If they are not all on the same host, network traffic will be required. Also, if the computation is not done on the user's host, the results will have to be shipped to him. Because network traffic puts a load on hosts and is quite slow compared to disk-memory transfers, total "cost" is probably very sensitive to this parameter.

Dynamic allocation of relations. The original placement of a relation and its backups is based on political and security considerations and on estimates of the impact on the host's load and network traffic. After experience is gained with the relation, it may become desirable to move it to another host. Essentially, the same parameters as discussed above are involved in deciding where to move it. However, now the DMS can supply information on what other relations are used with the relation to evaluate a query and where results are placed. The DBA can now choose the best site for the relation based on actual usage patterns rather than estimates.

Availability of a data base. The availability of a data base is a function of the MTBF and MTTR of the host systems, the DMS, and communication lines to the hosts. Besides adding extra communications lines, the only way the DBA can improve the availability of the data base is to add backup copies on other sites in the network. The DMS then has the added responsibility of maintaining the backups as images of the primary copy.

The DMS has to ensure that the backups receive all updates. This increases the local load, network traffic, and the load on the backup sites. As the number of backups increases, the real time required to keep them synchronized increases. It is hoped that the overhead required to maintain the backups will be offset by performance improvements resulting from altering the structure of the backups. The backups could be structured to be very efficient for classes of queries handled poorly by the primary copy. As long as the backup copy contained the same information as the primary, the availability would still be enhanced.

Clustering. The idea here is to store the data so that consecutively read records are often in the same block. Take, for example, a personnel file sorted on social security numbers where five records make a disk block. If a listing of the personnel file sorted on social security number is requested, the minimum amount of I/O will be used. However, if the listing is to be sorted on name, up to five times as much I/O would be required. It is up to the DBA to decide on which domain, if any, to have the relation sorted. An alternative discussed below is to used an index to "remember" a sort.

There are four parameters the DBA must evaluate when deciding how to order a relation.

1. <u>Cost of a sort</u>. If there are n tuples in a relation, the cost of sorting it will be O(n log n) and a substantial amount of temporary storage will be required for the duration of the sort. Also, no other users will be able to access the relation while it is being sorted.
2. <u>Number and types of queries</u>. Some queries will be much easier to evaluate when the relation is sorted because the relation is clustered properly for that query. However, other queries might run slower due to inefficient use of I/O channels. The DBA needs to understand the query load to be able to forecast the effects of a sort on the processing of queries.
3. <u>Number and types of updates</u>. It could be quite expensive to insert a tuple in the middle of a relation to keep it sorted. Or in the example above, given a social security number, it would be quite cheap to change the name field.
4. <u>Cost of storage</u>. The DBA should have the option of storing the relation at a backup site sorted on a different domain. Now a higher percentage of the queries will be cheap to answer but the overall cost of updates will increase.

<u>Index Selection</u>. The idea behind an index is to remember part or all of the tuples resulting from a query. In the example in the previous section, indices could be maintained for those domains on which the output is sometimes sorted. Considerable I/O will still be required

for those queries but much less than the cost of a sort. The DBA has to consider this savings as opposed to the increased cost of updates and the cost of storage when deciding what indices to add.

Views. If a query were asked often enough, it would become reasonable to save the answer to the query in a view, reducing the cost of that query the next time it was asked. Unfortunately, the cost of reflecting an update to one of the relations on which the view is based could be quite expensive. The cost would probably be a function of the number of joins involved in the query. Experience has to be gained with this tool before it can be properly evaluated.

Research Affecting the Experimental System

Previously Described Topics

The purpose of this section is to give a concise summary of the research required to support the experimental system. It should be emphasized that the experimental system is not simply a software project. Many of the component areas of the system are major research topics in their own right. The Research Plan (CAC Doc. No. 164, JTSA Doc. No. 5510) discussed several of these topics in detail. For completeness, Table 2, a summary derived from the Research Plan, is included.

Topic	Research needed
Synchronization	Application of operating system and data base synchronization methods.
Name space management	Extension of name space to allow network-wide naming of logical entities.
Network file allocation	Algorithms for determining suitable locations for the files.
Multiple copy management and backup	Design and analysis of the basic algorithms.
Data compression	Application of existing techniques.
Data clustering	Design and analysis of fast and effective techniques.
Deadlock	Application of existing techniques.
Automated structure design	Analysis of data required and algorithms for automated data structure design.

Table 2

Research topics covered in the Research Plan

There is another group of topics that is more specific to the particular approach we have taken. A discussion of these topics follows.

Relational Operators Suitable for Network Use

Research needed: design and implementation of relational operators better suited to network environments. The current definitions of operators for the relational algebra rely upon the existence of two complete relations. In a network environment where effective communication rates are on the order of 20-40 Kbits/sec, it could potentially take a very long time to send a complete relation to another host. The thrust of this area of research will be to define a relationally complete set of operators which will allow the same operations to proceed before a complete relation is transmitted. For example, our initial studies suggest that by sorting the relations in a join on the joining domain, the resulting join relation can be produced a tuple at a time. To do an unordered join requires O(m*n) work where m and n are the number of tuples in the two joining relations. A sort on a relation of k tuples requires O(k log k) work. A join in which both relations are ordered requires, in the normal case, O(m + n) work, i.e. a simple pass through each.

> Imagine two relations A and B which are both ascending sorted on a domain. To join them on that domain we chose one, say A, look at the top tuple and compare the values of the joining domain with the corresponding domain in the top tuple of B. If they are equal, the concatenated tuple is in the join, and should be output. If A is less, that tuple of A cannot possibly be in the join, so we discard it, making the next tuple of A the top (i.e. pop one tuple of A). If top of A is greater than top of B, pop B.

The join can be produced with O(m log m + n log n + m + n) work, with partial, intermediate results being available. Considerable analysis should be performed before we can be assured that such a "streamed" join is really suitable, but this is the direction the research should take.

Views and Inverse Views

Research needed: analysis of techniques for implementation. As discussed above, the concept of a view is an attempt to provide the user with a better interface to the system. By means of a view the user can have a data base that is very well suited to his own particular needs. Views may be used for access control by deleting from the user's view anything he is not permitted to see. The most important area of research on views is how they may be mechanized. In some research systems views are created by processing each query through what is essentially a generalized macro processor. In principle, the distributed DMS environment allows views to be physically created. This is because a distributed DMS is designed to handle multiple copies of relations. Physical creation of the view may be undesirable from a practical standpoint, since large amounts of storage may be required to support the views. There are also some theoretical problems regarding update anomalies. It is just not possible to perform certain classes of updates to a view. These anomalies occur when the underlying relations are such that no combination of changes to them will result in the desired change to the view.

Since the notion of an inverse view is a new one, it needs to be explored more completely. The issues are mainly in the area of management. If the inverse views are disjoint, the situation is simplified. However, there still may be problems in directing updates and queries to the proper host. When the inverse views are not disjoint, i.e. some tuples are represented in two or more inverse views, one faces the problem of update synchronization discussed throughout this document. It may well turn out that the notion of an inverse view is superfluous or inadequate. Research is needed to explore the limits of the concept.

Dynamic Optimization of Retrieval Strategies

Research needed: design of techniques for the optimization of both global (multiple relation) and local (within a single relation) retrieval strategies. We have presented a very brief discussion of some of the global optimization techniques in the section on the network data manager. A similar, brief discussion on local optimization is found in the section on the local data manager. The research needed is to perform a more thorough analysis of the possible techniques for retrieval strategy optimization, and to apply these results to the experimental system. Since the optimization must be performed on a dynamic basis for each query, it must operate quickly and effectively, lest the optimization cost more than it saves.

Implementation Considerations

Manpower

The initial design and implementation of the system will require approximately 50 man-months of effort over the course of the 13.3 month follow-on contract. Four different skill levels are required. The principal investigator should be a seasoned researcher who can direct the progress of the overall research program. The principal investigator will devote approximately half of his time to the experimental system effort and half his time to the complementary modeling effort. The second skill level required is a senior investigator. The senior investigator will be responsible for the coordination and management of the activities making up the experimental system effort. He should be capable of independent, creative research, as well as assuming management responsibilities for the experimental system design and implementation. A senior investigator will be devoted full time to the experimental system. The third skill level is systems analyst. Systems analysts should have broad competence in the design and implementation of software systems. Strong expertise in at least one theoretical or empirical area is necessary. Between two and three full-time systems analysts will be required. The fourth skill level is graduate research assistants. They should be Ph.D. or Masters candidates in computer science. One or two graduate research assistants will be utilized.

Schedule

The initial phases of the experimental system effort will be devoted to the design of the system. As portions of the design are finalized, implementation will begin. The design effort will commence immediately. A complete, though preliminary, system design should exist around the first of the year, i.e. four to five months into the contract. This design will be circulated for comments. An important component of the system design will be the specification and analysis of the basic algorithms for managing a multiple-copy, distributed DMS. Actual implementation of the system will begin fairly early, around two to three months into the contract. In these early phases, implementation will be at a low level of activity. Shortly after the completion of the design, implementation will become the dominant activity, and will remain so through the end of the contract.

In a previous section we discussed the impact of the other research activities on the experimental system. The estimated timings in this section could be substantially altered by major difficulties or breakthroughs on any of these research topics.

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CAC Document Number 170 JTSA Document Number 5512	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Research in Network Data Management and Resource Sharing - Preliminary Experimental System Design Report		5. TYPE OF REPORT & PERIOD COVERED Research Report - Interim
		6. PERFORMING ORG. REPORT NUMBER CAC #170
7. AUTHOR(s) Steve R. Bunch David C. Healy Edwin J. McCauley		8. CONTRACT OR GRANT NUMBER(s) DCA100-75-C-0021
9. PERFORMING ORGANIZATION NAME AND ADDRESS Center for Advanced Computation University of Illinois at Urbana-Champaign Urbana, Illinois 61801		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Joint Technical Support Activity 11440 Isaac Newton Square, North Reston, Virginia 22090		12. REPORT DATE August 20, 1975
		13. NUMBER OF PAGES 67
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Copies may be requested from the address given in (11) above.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) No restriction on distribution.		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Distributed data management Relational model Data management		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The basic design of an experimental distributed data management system is presented. The system is based upon the relational model.		